#include "Listaclubes.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Definición de Tipos de Datos para manejo interno \*/

/\*--------------------------------------------------\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Funciones Adicionales \*/

/\*-----------------------\*/

/\*

pre : ninguna.

post: compara ambos club1 y club2, devuelve

mayorclub si club1 es mayor que club2,

igualclub si club1 es igual a club2,

menor si club1 es menor que club2.

club1 : dato a comparar.

club2 : dato a comparar.

return resultado de comparar club1 respecto de club2.

\*/

ResultadoComparacionClubes compararDatoClubes(Clubes club1, Clubes club2) {

if (getId(club1) > getId(club2)) {

return MAYORCLUB;

}

else if (getId(club1) < getId(club2)) {

return MENORCLUB;

}

else {

return IGUALCLUB;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Implementación de Primitivas \*/

/\*------------------------------\*/

void crearListaClubes(ListaClubes &lista) {

lista.primero = finClubes();

}

/\*----------------------------------------------------------------------------\*/

bool listaVacia(ListaClubes &lista) {

return (primero(lista) == finClubes());

}

/\*----------------------------------------------------------------------------\*/

PtrNodoListaClubes finClubes() {

return NULL;

}

/\*----------------------------------------------------------------------------\*/

PtrNodoListaClubes primero(ListaClubes &lista) {

return lista.primero;

}

/\*----------------------------------------------------------------------------\*/

PtrNodoListaClubes siguiente(ListaClubes &lista, PtrNodoListaClubes ptrNodo) {

/\* verifica si la lista está vacia o si ptrNodo es el último \*/

if ((! listaVacia(lista)) && (ptrNodo != finClubes()))

return ptrNodo->sgte;

else

return finClubes();

}

/\*----------------------------------------------------------------------------\*/

PtrNodoListaClubes anterior(ListaClubes &lista, PtrNodoListaClubes ptrNodo) {

PtrNodoListaClubes ptrPrevio = finClubes();

PtrNodoListaClubes ptrCursor = primero(lista);

while (( ptrCursor != finClubes()) && (ptrCursor != ptrNodo)) {

ptrPrevio = ptrCursor;

ptrCursor = siguiente(lista,ptrCursor);

}

return ptrPrevio;

}

/\*----------------------------------------------------------------------------\*/

PtrNodoListaClubes ultimo(ListaClubes &lista) {

/\* el último nodo de la lista es el anterior al fin() \*/

return anterior(lista,finClubes());

}

/\*----------------------------------------------------------------------------\*/

PtrNodoListaClubes crearNodoLista(Clubes clubes) {

/\* reserva memoria para el nodo y luego completa sus datos \*/

PtrNodoListaClubes ptrAux = new NodoListaClubes;

ptrAux->clubes = clubes;

ptrAux->sgte = finClubes();

return ptrAux;

}

/\*----------------------------------------------------------------------------\*/

PtrNodoListaClubes adicionarPrincipio(ListaClubes &lista, Clubes clubes) {

/\* crea el nodo \*/

PtrNodoListaClubes ptrNuevoNodo = crearNodoLista(clubes);

/\* lo incorpora al principio de la lista \*/

ptrNuevoNodo->sgte = lista.primero;

lista.primero = ptrNuevoNodo;

return ptrNuevoNodo;

}

/\*----------------------------------------------------------------------------\*/

PtrNodoListaClubes adicionarDespues(ListaClubes &lista, Clubes clubes, PtrNodoListaClubes ptrNodo) {

PtrNodoListaClubes ptrNuevoNodo = finClubes();

/\* si la lista está vacia se adiciona la principio \*/

if (listaVacia(lista))

ptrNuevoNodo = adicionarPrincipio(lista,clubes);

else {

if (ptrNodo != finClubes()) {

/\* crea el nodo y lo intercala en la lista \*/

ptrNuevoNodo = crearNodoLista(clubes);

ptrNuevoNodo->sgte = ptrNodo->sgte;

ptrNodo->sgte = ptrNuevoNodo;

}

}

return ptrNuevoNodo;

}

/\*----------------------------------------------------------------------------\*/

PtrNodoListaClubes adicionarFinal(ListaClubes &lista, Clubes clubes) {

/\* adiciona el dato después del último nodo de la lista \*/

return adicionarDespues(lista,clubes,ultimo(lista));

}

/\*----------------------------------------------------------------------------\*/

PtrNodoListaClubes adicionarAntes(ListaClubes &lista, Clubes clubes, PtrNodoListaClubes ptrNodo) {

PtrNodoListaClubes ptrNuevoNodo = finClubes();

if (! listaVacia(lista)) {

if (ptrNodo != primero(lista))

ptrNuevoNodo = adicionarDespues(lista,clubes,anterior(lista,ptrNodo));

else

ptrNuevoNodo = adicionarPrincipio(lista,clubes);

}

return ptrNuevoNodo;

}

/\*----------------------------------------------------------------------------\*/

void colocarDato(ListaClubes &lista, Clubes &clubes, PtrNodoListaClubes ptrNodo) {

if ( (! listaVacia(lista)) && (ptrNodo != finClubes()))

ptrNodo->clubes = clubes;

}

/\*----------------------------------------------------------------------------\*/

void obtenerDato(ListaClubes &lista, Clubes &clubes, PtrNodoListaClubes ptrNodo) {

if ((! listaVacia(lista)) && (ptrNodo != finClubes()))

clubes = ptrNodo->clubes;

}

/\*----------------------------------------------------------------------------\*/

void eliminarNodo(ListaClubes &lista, PtrNodoListaClubes ptrNodo) {

PtrNodoListaClubes ptrPrevio;

/\* verifica que la lista no esté vacia y que nodo no sea fin\*/

if ((! listaVacia(lista)) && (ptrNodo != finClubes())) {

if (ptrNodo == primero(lista))

lista.primero = siguiente(lista,primero(lista));

else {

ptrPrevio = anterior( lista , ptrNodo );

ptrPrevio->sgte = ptrNodo->sgte;

}

// Si el dato es un TDA, acá habría que llamar al destructor.

delete ptrNodo;

}

}

/\*----------------------------------------------------------------------------\*/

void eliminarNodoPrimero(ListaClubes&lista) {

if (! listaVacia(lista))

eliminarNodo(lista,primero(lista));

}

/\*----------------------------------------------------------------------------\*/

void eliminarNodoUltimo(ListaClubes &lista) {

if (! listaVacia(lista))

eliminarNodo(lista,ultimo(lista));

}

/\*----------------------------------------------------------------------------\*/

void eliminarLista(ListaClubes &lista) {

/\* retira uno a uno los nodos de la lista \*/

while (! listaVacia(lista))

eliminarNodo(lista,primero(lista));

}

/\*----------------------------------------------------------------------------\*/

PtrNodoListaClubes localizarDato(ListaClubes&lista, Clubes clubes) {

bool encontrado = false;

Clubes clubesCursor;

PtrNodoListaClubes ptrCursor = primero(lista);

/\* recorre los nodos hasta llegar al último o hasta

encontrar el nodo buscado \*/

while ((ptrCursor != finClubes()) && (! encontrado)) {

/\* obtiene el dato del nodo y lo compara \*/

obtenerDato(lista,clubesCursor,ptrCursor);

if (compararDatoClubes(clubesCursor,clubes) == IGUALCLUB)

encontrado = true;

else

ptrCursor = siguiente(lista,ptrCursor);

}

/\* si no lo encontró devuelve fin \*/

if (! encontrado)

ptrCursor = finClubes();

return ptrCursor;

}

/\*----------------------------------------------------------------------------\*/

void eliminarDato(ListaClubes &lista, Clubes clubes) {

/\* localiza el dato y luego lo elimina \*/

PtrNodoListaClubes ptrNodo = localizarDato(lista,clubes);

if (ptrNodo != finClubes())

eliminarNodo(lista,ptrNodo);

}

/\*----------------------------------------------------------------------------\*/

PtrNodoListaClubes insertarDato(ListaClubes &lista, Clubes clubes) {

PtrNodoListaClubes ptrPrevio = primero(lista);

PtrNodoListaClubes ptrCursor = primero(lista);

PtrNodoListaClubes ptrNuevoNodo;

Clubes clubCursor;

bool ubicado = false;

/\* recorre la lista buscando el lugar de la inserción \*/

while ((ptrCursor != finClubes()) && (! ubicado)) {

obtenerDato(lista,clubCursor,ptrCursor);

if (compararDatoClubes(clubCursor,clubes) == MAYORCLUB)

ubicado = true;

else {

ptrPrevio = ptrCursor;

ptrCursor = siguiente(lista,ptrCursor);

}

}

if (ptrCursor == primero(lista))

ptrNuevoNodo = adicionarPrincipio(lista,clubes);

else

ptrNuevoNodo = adicionarDespues(lista,clubes,ptrPrevio);

return ptrNuevoNodo;

}

/\*----------------------------------------------------------------------------\*/

void reordenar(ListaClubes &lista) {

ListaClubes temp = lista;

PtrNodoListaClubes ptrCursor = primero(temp);

crearListaClubes(lista);

while ( ptrCursor != finClubes() ) {

Clubes clubes;

obtenerDato( temp, clubes, ptrCursor);

insertarDato( lista, clubes );

eliminarNodo( temp, ptrCursor );

ptrCursor = primero(temp);

}

eliminarLista( temp );

}

/\*----------------------------------------------------------------------------\*/

int longitud(ListaClubes &lista){

PtrNodoListaClubes ptrCursor = primero(lista);

int longitud = 0;

while ( ptrCursor != finClubes() ) {

longitud++;

ptrCursor = siguiente( lista, ptrCursor);

}

return longitud;

}

Clubes traerClub(ListaClubes clubes, int id){

Clubes club;

PtrNodoListaClubes cursorclub;

cursorclub=primero(clubes);

while(cursorclub!=finClubes()){

obtenerDato(clubes,club,cursorclub);

if(id==getId(club)){

cursorclub=finClubes();

}else{

cursorclub=siguiente(clubes,cursorclub);

}

}

return club;

}

/\*----------------------------------------------------------------------------\*/